Journal of Advances in Science and Technology

Vol. IV, Issue No. VII, November-2012, ISSN 2230-9659

A STUDY ON IMPLEMENTATION OF IMAGE PROCESSING FOR MULTIPLICATION IMAGE PROCESSING APPLICATION

AN INTERNATIONALLY INDEXED PEER REVIEWED & REFEREED JOURNAL

www.ignited.in

Journal of Advances in Science and Technology
Vol. IV, Issue No. VII, November-2012, ISSN 2230-9659

A Study on Implementation of Image Processing for Multiplication Image Processing Application

K. Seetharam

Research Scholar, CMJ University, Shillong, Meghalaya

Abstract – Multiplication is the kernel operation used in many image and signal processing applications. In this paper, we present the design and (EPGA) implementation of multiplier architectures for use in image and signal processing applications. The designs are optimized for speed which is the main requirement in these applications. First design involves computation of dense multiplication which is used in image processing application. The design has been implemented on Virtex-4 EPGA and the performance is evaluated by computing the execution time on EPGA. Implementation results demonstrate that it can provide a throughput of 16970 frames per second which is quite adequate for most image processing applications. The second design involves multiplication of tri-matrix (three matrices) which is used in signal processing application. The proposed design for the multiplication of three matrices has been implemented on Spartan-3 and Virtex-II Pro platform EPGAs respectively. Implementation results are presented which demonstrate the suitability of EPGAs for such applications.

Keywords: EPGA, Implementation, Image Processing, Multiplication, Processing, Application.

----------♦----------

INTRODUCTION

Computation intensive algorithms used in image and signal processing, multimedia, telecommunications, cryptography, networking and computation domains in general were first realized using software running on Digital Signal Processors (DSPs) or General Purpose Processors (GPPs). Significant speed-up in computation time can be achieved by assigning complex computation intensive tasks to hardware and by exploiting the parallelism in algorithms (Ogrenci, *et. al.,* 2003). Recently, (EPGAs) have become a platform of choice for hardware realization of computation-intensive applications (Ogrenci, *et. al.,* 2003. Ebeling, *et. al.,* 2004. Goslin, 1997. Isoaho, *et. al.,* 1993. Ye and Lewis, 1999. Knapp. Ma, 2003. Otto and Pavel, 2002. Batina, *et. al.,* 2003. Johnson, *et. al.,* 2002. Compton and Hauck, 2002. Tessier and Burleson, 2001. Todman, *et. al.,* 2005. Especially, when the design at hand requires very high performance, designers can benefit from high density and high performance EPGAs instead of costly multicore Digital Signal Processing (DSP) systems (Ogrenci, *et. al.,* 2003). EPGAs enable a high degree of parallelism and can achieve orders of magnitude speedup over GPPs (Ma, 2003). This is as a result of the increasing embedded resources on EPGA. EPGA have the benefits of the hardware speed and the software flexibility; also they have a price/performance ratio much more favorable than Application Specific Integrated Circuits (ASICs). Since the major resources for implementing computation-intensive algorithms are embedded on EPGA, latency associated with device communication has been eliminated. However, these embedded resources are limited hence it is important to use these resources optimally. The last decade has seen ever increasing application areas for EPGAs. Modern EPGAs currently accommodate more than ten million gates with clock rates approaching 550 MHz (Todman, *et. al.,* 2005). Example application areas include single chip replacements for old multichip technology designs, DSP, image processing, multimedia applications, high-speed communications and networking equipment such as routers and switches, the implementation of bus protocols such as Peripheral Component Interconnect (PCI), microprocessor glue logic, coprocessors and controllers. Most of the computation intensive algorithms such as those used in signal, image and video processing, numerical analysis, computer graphics and vision involve matrix operation as the kernel operation. In this paper, different architectures of matrix multiplication for use in image and signal processing applications are considered for hardware realization using EPGA.

REVIEW OF LITERATURE

Multiplication is a computationally intensive problem, especially the design and efficient implementation on an EPGA where resources are very limited, has been more demanding. EPGA based designs are usually evaluated using three performance metrics: speed (latency), area, and power (energy). Fixed point

implementations in EPGA are fast and have minimal power consumption. Additionally, a fixed point matrix multiplier unit often requires less silicon real estate in an EPGA or ASIC than its floating-point counterpart. The limitation of fixed point number is that very large and very small numbers cannot be represented and the range is limited to bit-width of the number. There has been extensive previous work in the area of designing an EPGA based system for the computation. The authors of used multiplication as the benchmark to compare the performance of EPGAs, DSPs and embedded processors. The results show that the EPGAs can multiply two matrices with both lower latency and lower energy consumption than the other two types of devices. This makes EPGA ideal choice for matrix multiplication in signal processing applications.

EPGA Overview

Programmable devices, such as programmable logic arrays (PLAs), have been available since 1970s. However, for a number of years, their use was quite limited, mainly due to technological reasons. In the early 1980s, programmable array logic (PALs) devices started to be used as glue-logic parts but suffered from power consumption problems. The extension of the gate array technique to post manufacturing customization, based on the idea of using arrays of custom logic blocks (LBs) that are surrounded by a perimeter of input/output (I/O) blocks, all of which could be assembled arbitrarily (Rodriguez-Andina, *et. al.,* 2007, Pellerin and Thibault, 2005) gave rise to the EPGA concept, which was introduced by Xilinx' cofounder Ross Freeman in 1985. EPGAs are digital integrated circuits (ICs) that belong to a family of programmable logic devices (PLDs). An EPGA chip includes I/O blocks and the core programmable fabric. The I/O blocks are located around the periphery of the chip, providing programmable I/O connections and support for various I/O standards. The core programmable fabric consists of programmable logic blocks also called configurable logic blocks (CLBs) and programmable routing architectures. By using the appropriate configuration, EPGAs can, in principle, implement any digital circuit as long as their available resources are adequate. Fig. 1 illustrates a general EPGA fabric (Maxfield, 2004) which represents a popular architecture that many commercial EPGAs are based on, and is also a widely accepted architecture model used by EPGA researchers.

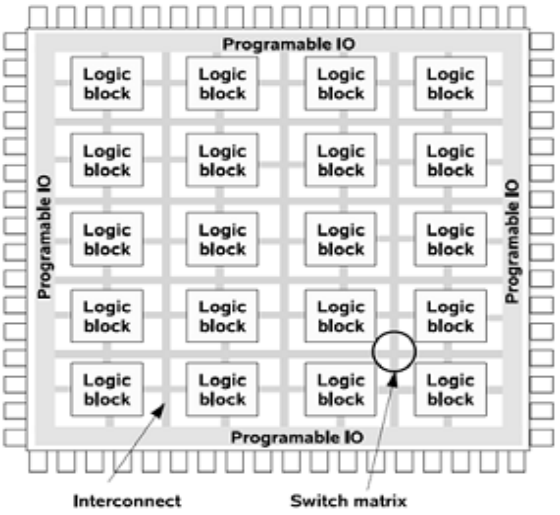

Fig. 1 General EPGA fabric

EPGAs can be programmed after it is manufactured rather being limited to a predetermined, unchangeable hardware function. The term "field programmable" refers to the fact that its programming takes place "in the field" as opposed to devices whose internal functionality is hardwired by the manufacturer. Many different architecture and programming technologies have evolved to provide better designs that make EPGAs economically viable and an attractive alternative to ASICs. Modern EPGAs have superior logic density, low chip cost and performance specifications comparable to low end microprocessor. With multimillion programmable gates per chip, current EPGAs can be used to implement digital systems capable of operating at frequencies up to 550 MHz. In many cases, it is possible to implement an entire system using a single EPGA. This is very economical for specialized applications that do not require the performance of custom hardware. Significant technological advancements have led to architectures that combine EPGA's logic blocks and interconnect matrices, with one or more microprocessors, embedded Intellectual Property (IP) cores, memory blocks, DSP blocks integrated on a single chip to facilitate the implementation of Programmable System-on-a-Chip (PSoC) designs. Examples of PSoC are the Xilinx Virtex-II Pro, Virtex-4 and Virtex-5 EPGA families, which include one or more hard-core PowerPC processors embedded along with the EPGA's logic fabric. Alternatively, soft processor cores that are implemented using part of the EPGA logic fabric are also available. Many soft processor cores are now available such as: Xilinx 32-bit Micro Blaze and Pico Blaze, and the Altera Nios and the 32-bit Nios II processor (Rodriguez-Andina, *et. al.,* 2007).

COMPARISON OF EPGA$_S$ WITH ASIC$_S$, GPP$_S$ AND DSP$_S$

An ASIC is highly optimized for one specific application or product. ASICs can provide the best performance and lowest power consumption. For

large volume applications, ASICs can also provide the lowest chip cost and system cost. Despite the advantages of ASICs, they are often infeasible or uneconomical for many embedded systems because of high nonrecurring engineering (NRE) cost and longer design time (Todman, *et. al.,* 2005. Rodriguez-Andina, *et. al.,* 2007). As compared to ASICs, EPGAs offer many advantages such as reduced NRE cost and shorter time to market. However, relatively high size and power consumption shown by EPGA devices has been the most important drawback of that technology. GPPs on the other hand are microprocessors that are designed to perform a wide range of computing tasks. As mentioned earlier, EPGAs are most often contrasted with ASICs. However, before deciding on the implementation technology, it is very important to study the application carefully and then determine if it is possible to meet performance requirements with existing programmable processors-GPPs or DSPs. Development of code for such processors require much less effort as compared to that required for EPGAs or ASICs, because developing software with sequential languages such as C or C++ is much less challenging than writing parallel code with Hardware Description Languages (HDLs) GPPs are also generally cheaper than EPGAs. Hence, if a GPP can meet application requirements (performance, power, etc.), it is almost always the best choice. In general, EPGAs are well suited to applications that demand extremely high performance and reprogram ability. Power consumption in a DSP depends on the number of memory elements used regardless of the size of the executable program. For EPGA, the power consumption depends on the circuit design. EPGAs are important when there is a need to implement a parallel algorithm, that is, when different components operate in parallel to implement the system functionality. Thus the speed of execution is independent of the number of modules. This is in contrast to DSP systems where the execution speed is inversely proportional to the number of functionalities. EPGAs deliver an order of magnitude higher performance than DSPs.

METHODOLOGY

Design methodology for the hardware realization of computation intensive algorithm is a combined effort of Electronic Design Automation (EDA) tools, methods and EPGA technology that enables to produce the optimized circuit for the end applications. A right combination of EPGA hardware, designed IP core and EDA tools will definitely enhance the efficiency of the design methodology. By design methodology, we imply the step-by-step process of EPGA design. The EPGA design methodology is used as a guideline for the hardware realization of algorithms. A number of design flows are used by different EPGA vendors but all are basically similar in sequence of tasks performed. These steps are common in all EPGA EDA tools and are essential in today's EPGA design process. The EDA tools like Xilinx Integrated Software Environment (ISE), Altera's Quartus II and Mentor Graphics' EPGA Advantage plays a very important role in obtaining an optimized digital circuit using EPGA (Todman, *et. al.,* 2005. Rodriguez-Andina, *et. al.,* 2007). A typical EPGA design flow followed in this work is shown in fig. 2. In this flow, design Entry is used to describe the algorithm/circuit that has to be implemented onto the EPGA device. There are two standard approaches to specify the EPGA designs: HDL-based and Schematic based depending upon the complexity of EPGA design. However, for complex and computationally intensive algorithms HDL-based (VHDL or Verilog) design entry is the dominant method used by EPGA designers. After specifying the design using HDLs or Schematic, the designer needs to validate the logical correctness of the design. This is performed using functional or behavioral simulation.

Designers usually go through this step right after they finish the design entry and logic synthesis. Logic synthesis converts HDL or schematic-based design entry into a net list of actual gates/blocks specified in EPGA devices. This is the most important step of the whole design process. Technology mapping is a step in the middle of typical EPGA design flow. In this step, the EDA tool transforms a net list of technology independent logic gates into one comprised of logic cells and IOBs in the target EPGA architectures. Mapping plays a significant role on the quality of the implemented circuits.

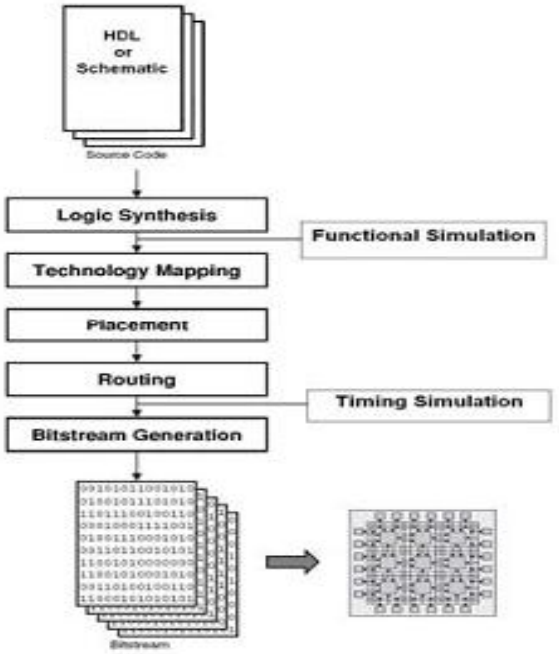

Fig. 2 EPGA designs flow.

Placement follows technology mapping in an EPGA design flow and selects the optimal position for each block in a circuit. A good placement is extremely important for EPGA designs. It directly affects the rout ability and the performance of a design on EPGA. EPGA placement algorithms can be broadly classified as rout ability-driven and timing-driven. The next step in the EPGA design flow is routing. It is the last step in the design flow prior to generating the bit-stream to program the EPGA. Analysis is essential for today's designs that have complex algorithms and huge amount of gates. The analysis tools (Model Sim, ISE simulator, Quartus II) are linked with the initial step and when an error occurs, the whole design has to go back to previous steps or in certain situations to the very beginning depending on the severity of the problem. Timing simulation validates the logical correctness of the design taking into account the delays of the EPGA device. Bit stream generation and downloading the generated bit file in the EPGA is the final step of the EPGA design flow.

IMPLEMENTATION

EPGA-based systolic array parallel architecture for the trimatrix multiplication was evaluated for different matrix sizes ranging from 3×3 to 7×7 multiplications. As shown in table 1, roughly 14% of the slices and 57% DSP48s are utilized leaving a plenty of room to implement more parallel processors on the same EPGA chip. The results listed in table 2 were obtained using Xilinx ISE 9.2i tool configured to optimize for speed. The total processing time using Virtex-4 EPGA is found to be 58.93 µs; this is equivalent to a throughput of 16970 frames per second. The results indicate the feasibility of using EPGA for real time high speed image processing applications using this matrix-vector multiplication.

Table 1: EPGA Resource Utilization

Resource	*Used/Available*	*Utilization*
Slices	1,3010 out of 89,088	14%
4-input LUTs	9,612 out of 178,176	5%
DSP48s	55 out of 96	57%
Max. Frequency	17.376 (MHz)	-

EPGA-based systolic array parallel architecture for the trimatrix multiplication was evaluated for different matrix sizes ranging from 3×3 to 7×7 tri-matrix multiplications. The same architecture is extended for 7×7 tri-matrix multiplier. For 7×7 tri-matrix multiplier, the EPGA utilizes more resources as compared to 3×3 tri-matrix multiplier. It requires more hardware resources which is obvious from the computational complexity of 7×7 multiplier. The implementation results are summarized and compared in table 2. Since the 7×7 design could not be fit into Spartan- 3 (XC3S2000FG900-4) device, we used a more advanced Virtex-II Pro (XC2VP100FF1704-6) platform EPGA for implementation.

Table 2: EPGA Resource Utilization Comparison

EPGA Resources	*T=[ ]7×7 (Virtex-II Pro)*	*T=[ ]3×3 (Spartan-3)*
LUTs	2,353	270
CLB Slices	1,177	144
Eq. Gate Count	1,151,761	148,215
Max. Frequency	102 MHz	87 MHz
Embedded Multipliers	280	36

CONCLUSIONS

Most of the algorithms which are used in DSP, image and video processing, computer graphics and vision and high performance supercomputing applications have multiplication as the kernel operation. In this paper, we considered two different examples of multiplier architecture where speed is the main constraint. The first design involving computation of dense matrix-vector multiplication is implemented on Xilinx Virtex-4 EPGA and the performance is evaluated by computing its execution time on EPGA. Hardware implementation results demonstrate that it can provide a throughput of 16970 frames per second which is sufficient for many image and video processing applications. The second design for the multiplication of three matrices is based on systolic array and implemented on Spartan-3 and Virtex-II Pro platform EPGAs respectively. Implementation results demonstrate the suitability of EPGAs in such applications. Finally, we conclude that for multiplication of large matrices, memory based architecture is quite efficient whereas, for small and medium sized matrix multiplication, systolic array techniques prove to be quite efficient as demonstrated by the implementation results.

REFERENCES

S. Ogrenci, A. K. Katsaggelos, and M. Sarrafzadeh, (2003). "Analysis and EPGA Implementation of Image restoration under resource constraint," IEEE Trans. on Computers, Vol. 52, No. 3, pp. 390-399.

C. Ebeling, C. Fisher, G. Xing, M. Shen, and H. Liu, (2004). "Implementing an OFDM Receiver on the RaPiD Reconfigurable Architecture," IEEE Trans. on Computers, Vol. 53, No. 11, pp. 1436-1448.

G. R. Goslin, (1997). "A Guide to Using Field Programmable Gate Arrays for Application-Specific Digital Signal Processing

Performance," Microelectronics Journal, Vol. 28, Issue 4, pp. 24-35.

J. Isoaho, J. Pasanen, O. Vainio, and H. Tenhunen, (1993). "DSP System Integration and Prototyping with EPGAs," Journal of VLSI Signal Processing, Vol. 6, pp. 155-172.

A. G. Ye and D. M. Lewis, (1999). "Procedural Texture Mapping on EPGAs," in Proc. of ACM/SIGDA 7th Intl. Symp. on Field Programmable Gate Arrays, pp. 112-120.

S. Knapp, "Using Programmable Logic to Accelerate DSP Functions," http://www.xilinx.com/appnotes/dspintro.pdf.

J. Ma, (2003). "Signal and Image processing via Reconfigurable Computing," in Proc. of the First Workshop on Information and Systems Technology.

F. Otto and Z. Pavel, (2002). "Hardware Accelerated Imaging Algorithms," in Proc. of AUTOS'2002 Automatizace systému, pp. 165-171.

L. Batina, S. B. Ors, B. Preneel, and J. Vandewalle, (2003). "Hardware architectures for public key cryptography," Integration, the VLSI Journal, Vol. 34, pp. 1-64.

D. Johnson, K. Gribbon, D. Bailey, and S. Demidenko, (2002). "Implementing Digital Signal Processing Algorithm's in EPGA's: Digital Spectral Warping," in Proc. of 9th Electronics New Zealand Conf., pp. 72-77.

K. Compton and S. Hauck, (2002). "Reconfigurable Computing: A Survey of Systems and Software," ACM Computing Surveys, Vol. 34, No. 2, pp.171-210.

R. Tessier and W. Burleson, (2001). "Reconfigurable Computing for Digital Signal Processing: A survey," Journal of VLSI Signal Processing, Vol. 28, No. 3, pp.7-27.

T. J. Todman, G. A. Constantinides, S. J. E. Wilton, O. Mencer, W. Luk, and P. Y. K. Cheung, (2005). "Reconfigurable Computing: architectures and design methods," IEE Proc. of Computer Digital Techniques, Vol. 152, No. 2, pp. 193-207.

J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, (2007). "Features, Design Tools and Application Domains of EPGAs," IEEE Trans. on Industrial Electronics, Vol. 54, No. 4, pp. 1810-1823.

D. Pellerin and S. Thibault, (2005). "Practical EPGA programming in C, Prentice Hall, New York, USA, First Edition.

C. M. Maxfield, (2004). "The Design Warrior's Guide to EPGAs, Elsevier Publishers, New York, USA, First Edition.